ECE 271 – Microcomputer Architecture and Applications Lecture 11

Vince Weaver

`http://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

26 February 2019

Announcements

- Read Chapters 7 and 8
- Midterm, likely 12 March (two weeks)
  more info on that as it gets closer

Lab #5 Update

- Simple bugs continue to be the biggest problem
  If you are stuck, ask a TA or me! Don't waste more than an hour or so if you are really stuck.
- Don't use floating point in the lab
  It might work, but we haven't learned about it yet.
- Multiply/divide ordering in C
  - steps=(512*degrees)/360;
  - steps=512*(degrees/360);
  - are the above equivalent? Mathematically, yes.

In C, no. When using 32-bit integers, a number like 270/360 is going to evaluate to "0.75" which C will truncate to "0", not giving the result you expect.

Character Encodings

- What makes a text character?
  - Our processor only understands binary.
  - The letter 'A' we say is 65 (0x40).
  - Is that implicit in the processor or in the nature of the letter 'A'?
  - No, it's arbitrary
  - Why have a standard like this? Otherwise it would be impossible to communicate text! Every computer would treat letters differently.

- ASCII – American Standard Code for Information Interchange
  - Standard from the 1960s
  - Nice features
  - Numbers are consecutive, from 0x30-0x39 (easy to convert to decimal)
  - Letters are consecutive
  - Lower case has constant offset from uppercase, easy conversion
  - Technically 7-bit. What do you do with 8-bit? Parity? Extended characters?

  - Also control chars in bottom. Things like BELL (control-G), linefeed, carriage return, escape, etc.
- EBCDIC – IBM's standard. There were others. Some put char in 6 bits.
- Old systems missing chars? Uppercase only? How did people cope? How did the C compiler cope? Trigraphs.
- What about non-English languages. ess-tset? Umlauts?
- Unicode? 16-bit?
  wchar_t? Windows? Java? Will all languages fit in 16-bits? no
- UTF-8?

Top bit 1 indicates more than 1-byte long, can encode in up to 4 bytes. Regular C string manipulation will work on UTF-8, 7-bit ASCII is a subset

- Combining chars, security aspect of letters that look the same
- Politics involved.
- Emojis?

Functions/Subroutines

- Why use them?

Sample C

```c
int sum(int a, int b) {
   return a+b;
}

int main(int argc, char **argv) {
   int result;
   result=sum(1,2);
}
```

Sample Assembly

```
    mov    r0,#1
    mov    r1,#2
    bl     sum

sum:
    add    r0,r1,r2
    blx    lr
```

Subroutines on ARM

- `bl` branch and link instruction
  - Sets the link register LR (r14) as the memory address of the next instruction immediately after the BL (PC+4 on Thumb-2)
  - Adjust the PC to be the memory address of the first location of where we want to transfer execution
- After executing, LR has the return address

Returning from a Subroutine

- Use the `BX LR` instruction, which says to branch to the address located in the LR register. (the X is for exchange; historical THUMB reasons)
- Alternately, if the LR register is on top of the stack, you can `POP {PC}`

```
; p164
```

The ABI – The Application Binary Interface

- A Document, produced often by the processor maker
- An agreement of how functions / code talk to each other
- A common standard so compilers, libraries, and code can call each other and know how to set things up
- Useful to have for your own code. Might be slightly less efficient, but better than for every function you call having to save/setup different registers
- What kinds of things are included?
  - What registers to put things in? Register allocation?

- Alignment of stack (4 bytes? 8 bytes?)
- How to pass 8/16/32/64 byte values
- How to pass floating point values
- Where does the return value go?
- System calls
- Frame pointer

ARM ABI

- On Linux there have been at least 4
- armbe – big endian
- armle – little endian
- EABI – extended (new) ABI
- armhf – EABI but fancier (hard) floating point support

Calling Conventions

- r0/r1/r2/r3
  - parameters/scratch
  - caller saved, so if you want the value in say r3 to be the same after a function call you have to save it/restore it to memory
  - r0/r1 also used as return value from function
- r4/r5/r6/r7/r8/r10/r11 = variables
  - callee saved. You can count on this having the same value after a function as before. If you are in a function

and want to use it, must save/restore it. Often this done at function entry/exit

- r9 – implementation dependent (thread-local register?)
- r12 = reserved by linker?
- r13 = stack pointer
- r14 = LR (link register)
- r15 = PC (program counter)

Calling Conventions – Corner Cases

- Return value in r0. Might be in r1 or more if bigger than 32 bits
- What happens if more than 4 arguments?
- What happens if more than 32-bits (use 2 registers, even/odd for 64-bits)

Calling Conventions – Corner Cases

- How do you pass something complicated like an array or struct?
- Call by value or by reference
- Can pass a pointer in a 32-bit register